Docket Number: POU920000165US1

Inventor: S. Kinder et al.

Title: A Method for Creating Path-Sensitive Branch Registry for Cyclic Distributed Transactions

APPLICATION FOR UNITED STATES

LETTERS PATENT

"Express Mail" Mailing Label No.: EK830786344US
Date of Deposit: December 21, 2000

I hereby certify that this paper is being deposited with the United States Postal Service as "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to Box Patent Application, Assistant Commissioner for Patents, Washington, DC 20231.

Name: Karen L. Merrigan

Signature: Karen L. Merrigan

INTERNATIONAL BUSINESS MACHINES CORPORATION

A METHOD FOR CREATING PATH-SENSITIVE BRANCH REGISTRY FOR CYCLIC DISTRIBUTED TRANSACTIONS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to a method for creating a path-sensitive branch registry for use in a processing system and, more specifically, to identifying branch flows as subordinate relative to a path in the distributed transaction tree.

2. Description of the Related Art

Currently, the method of managing distributed transactions used in computer systems includes a system of tree structures having a plurality of processing nodes which are logically connected. Normally, each node's transaction manager automatically increments its present identifier at the end of processing each transaction to derive the next transaction identifier. When a distributed transaction is forced into a consistent state its associated superior/subordinate relationship becomes fixed. Typically, a transaction identifier is incremented in each of the nodes of a superior transaction manager. Consequently, transaction tasks in the superior transaction manager then proceed with the incremented identifier to one or more subordinate transaction managers. Similarly, the subordinate nodes assign a static branch qualifier. However, the subordinate transaction manager's identifier is modified to conform with its superior. As is the case with transactions, superior nodes often times receive instructions from nodes indirectly from the superior's subordinates. This situation results in cyclic distributed transactions that contain at least one loopback. The current practice is unable to ensure the safe creation of

cyclic distributed transactions, without worries of database update failures, or unwanted tightly coupled behavior.

In addition, typically, for modern object servers, the two phase commit process is preceded by a phase known as "before-completion." This process permits the object server to flush updates, cached in the object representation, to backing resource managers prior to the well-known two-phase commit process.

For distributed transaction trees that contain cycles, it is sometimes not possible for all updates to be pushed to the backing resource manager prior to that resource manager receiving the first part of the two-phase commit process. This will cause the resource manager to log an error, and commonly mark the global transaction rollback.

Therefore, there is a need for a method that unwinds the cyclical distributed transaction tree, while preserving the path for which the tree was allocated, regardless of the depth of the tree which maintains proper ordering of events and preventing unwanted sharing of resources.

SUMMARY OF THE INVENTION

An exemplary embodiment of the invention is a method for a path-sensitive branch registry for cyclic distributed transactions. This method has a superior node's transaction manager (TM) identifying itself as the root before sending syncpoint cues to at least one subordinate node. Before sending the syncpoint cues to the subordinate the superior links the cues with its specific branch qualifier (BQUAL) as well as a global transaction identifier (GTRID). The TM of each subordinate node receives syncpoint cues and is responsible for knowing who its superior is. In addition, the TM is responsible for recognizing the flow of branch instructions and guarantee that it uses a network-wide

unique value for the branch values it generates for a given global transaction. With the recognition of the flow from the superior node the subordinate TM updates the node registry as to the inbound and outbound flow of branch instructions by its superior and its subordinates.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings, wherein like elements are numbered alike in several FIGURES:

FIG. 1 is an exemplary diagram of a superior node and a subordinate node in the distribution transaction tree in one embodiment;

FIG. 2 is an exemplary diagram of a simple flow between nodes of the distribution transaction tree in one embodiment;

FIG. 3 is an exemplary diagram of a cyclical flow between nodes of the distribution transaction tree in one embodiment;

FIG. 4 is an exemplary diagram of a cyclical flow between nodes of the distribution transaction tree with path sensitivity in one embodiment;

FIG. 5 is an exemplary diagram of an acyclical flow between nodes of the distribution transaction tree with path sensitivity in one embodiment; and

FIG. 6 is an exemplary diagram that illustrates the interrelationship between superior and subordinate transaction managers in one embodiment.

DETAILED DESCRIPTION OF THE INVENTION

As discussed herein, currently cyclic distribution trees cannot ensure the safe creation of cyclic distributed transactions, without worries of database update failures. The current practice requires that when a cycle occurs in a distributed transaction tree that the resultant reentrancy does not cause new work to occur after the resource manager has been directed to prepare for commit by the TM.

FIG. 1 is an exemplary diagram depicting a typical flow from a superior or root node to a subordinate node. Specifically, a root node 12 is sending syncpoint cues (outbound messages) of the distributed transaction tree to subordinate nodes represented as 14, 18 and 22. The subordinate nodes 14, 18 and 22 each have a specific global transaction identifier (GTRID) which is the same in each node of the tree, and unique in the network for each distributed transaction tree. The GTRID is assigned by a root coordinator, and is propagated as the transaction flows from node to node. FIG. 1 demonstrates GTRID naming with BQUALs 16, 20 and 24. In addition, each node of a distributed transaction can be referred to as a branch, and also, within each node resides a transaction manager (TM). The TM manages a branch registry, recording inbound and outbound participants. Inbound flows represent flows from a superior node in the distributed transaction tree. Further, each transaction has a unique identifier that represents a singleton instance of a given node in the distributed transaction tree. These unique identifiers are received by the subordinate node's TM and are generally statically bound. Next, the incremented identifier is sent to another subordinate node that is able to coordinate and process the particular syncpoint cues. All node transactions utilize registries that are managed by the node's TM. With this registry the TM is able to track the responsibilities of the superior node.

FIG. 2 is a diagram depicting a directed acyclic distributed transaction tree. A distributed NT node 52 usually has a well-known name that it uses as its branch qualifier (BQUAL)

56, 60. The distributed NT node 52 sends syncpoint cues in the form of outbound flow 62 to a subordinate node 54. The outbound flow is received as inbound flow 62 by the subordinate node 54. The TM of the subordinate node 54 examines its registry for the inbound flow's GTRID before proceeding. If the TM of the subordinate node 54 has not seen the incoming flow's 62 GTRID it will create a new branch in the registry for this transaction, and add the BQUAL to the inbound branch registry. If the TM of subordinate node 54 has seen the incoming flow 62, it uses the transaction that was created previously to coordinate updates to protected resources. Thus, the TM considers this reentry as a direct synchronous inbound flow because the distributed NT node 52 may flow to subordinate node 54 and to subordinate node 58 repeatedly with no update to BQUAL 60 prior to committing.

FIG. 3 is a diagram depicting a distributed transaction tree containing a cycle. Node 104 receives two inbound flows 116, 118; one from the NT node 102, and one from node 114. When node 104 receives the flow there is the possibility that it will erroneously assume that this is the same branch in the distributed tree that was noted before. If this branch is assumed to be the same as the one previously sent there will be a likelihood that there will be an unwanted sharing of protected resource locks. Further, node 104 may become confused with its syncpoint responsibilities. To correct this confusion, node 104 will receive a prepare flow from the NT node 102 and node 114 during the commit processing phase. Consequently, node 104 will be directed to prepare, for which it will drive pre-prepare instructions for all local resource managers, and after preparing will flow a prepare signal to node 110. In sending the prepare signal node 104 also sends its non-incremented BQUAL 106. Similarly, node 110 issues pre-prepare instructions and prepares local resources. Next, node 110 directs the flow to node 114 along with its non-incremented BQUAL 108. During the preparing of resources for node 114, it is possible that updates may flow to node 104. Consequently, the pre-prepare instruction of node 114 causes updates on node 104 and its non-incremented BQUAL 106, which has

been previously prepared. Unfortunately, allowing updates on node 104 without incrementing the BQUAL 112 will invariably result in an error.

FIG. 4 is a diagram depicting a solution to the problem introduced by cyclic distribution transaction trees by introducing a path-sensitive branch registry. Here, node 154 receives an inbound flow 166 from node NT 152. Node 154 does not find the inbound flow 166 from node NT 152 in its inbound registry. Next, node 154 will associate an indexed, transaction-unique BQUAL 156 (A1) with the inbound flow, where the index (1) indicates the number of times that the transaction has looped through the node 154. Subsequently, node 154 will send a flow 168 and its BQUAL 156 (A1) to node 160. Next, node 160 will receive the inbound flow 168, and associate its own indexed BQUAL 158 (B1) with the inbound indexed BQUAL 156 (A1). Likewise, node 160 sends a flow 170 with its indexed BQUAL 158 (B1) to node 164. The cycle completes when node 164 sends a flow 172 with its indexed BQUAL 162 (C1) to node 154, where node 154 will consult its inbound registry to see that it has not received an inbound flow from node 164 for this transaction, and will create a new BQUAL (A2) with an incremented index (2) that is different for any other index in the registry for that node for that transaction. Therefore, the cyclic flow, NT → A → B → C → A has become the acyclic flow NT → A1 → B1 → C1 → A2.

FIG. 5 is a diagram depicting a creation of a path-sensitive registry. Node 202 is a subordinate of node 214 and other nodes based on the inbound flows 218. Node 202 is a superior to node 210. As in FIG. 3, NT will deliver a prepare instruction to node 202, in turn node 202 will issue pre-prepare instructions to the local resources. Node 202 will then prepare local resources, and flow prepare to node 210. Next, node 210 flows prepare to node 214. Consequently, after preparing local resources node 214 flows prepare to node 202. Following the prepare of the local resources node 202 then pre-prepares local resources associated with this subordinate transaction. Therefore, the path-sensitive registry prevents the unwanted sharing of database and/or protected resource locks and

correctly delivers pre-prepare to objects and prepare to resources. The unwanted sharing of database and/or protected resource locks and correct delivery of pre-prepare objects is achieved by incrementing BQUALs 204, 206, 208 and 212 before they are sent to another node.

FIG. 6 is a diagram depicting the relationship between transaction managers and their subordinates. As in FIG. 3, NT 252 represents a root transaction. The TM 254 of node 260 receives inbound flow 278 from the transaction root 252 and evaluated against the inbound registry of the node. Specifically, an inbound registry contains the node's BQUAL 258 and its GTRID 256. The TM 254 compares incoming syncpoint cues with those stored in the memory. When the TM 254 does not find a matching syncpoint cue it adds the incoming syncpoint cue to its registry and increments its BQUAL and links it to the syncpoint cue prior to sending it to its subordinate(s). In addition, the registry contains the node's BQUAL 258. For example, node 260 sends outbound flow to node 276 and node 274. The TM of a subordinate node 254 searches its registry to see if the inbound flow's GTRID matches any previously recorded GTRID. If there is no match the TM 254 directs the recording of the new GTRID. If there is a match the TM 254 renames the transaction and sends the newly named flow 282 to its subordinate(s), and so on. Next, the subordinate nodes 274 receives the outbound flow 282 and reports back to the superior node 260, confirming its subordinate status.

The embodiment described above solves the problems by introducing a method that unwinds the pretzel-like situations created by cyclic flows. This method ensures that transaction managers are able to properly drive the syncpoint cues with the proper superior or root TM in the transaction. In addition, the exemplary embodiment described above enables the safe creation of cyclic distributed transactions, free from database update failures.

The preferred embodiment uses a optimization that fully prepares the nodes at the same depth in the synchronization tree, and at completion of the prepare phase if there are distributed subordinate registrations for the related transactions. In addition, the superior nodes receive a prepare signal to initiate node registration. During the start of the prepare phase, the subordinate will drive locally registered synchronization objects before completion methods are run, and so on. Consequently, this optimization can cause the object server to attempt to drive "work after prepare" to the resource managers in the case of a cyclic tree. This application preserves the performance optimization while properly delivering the "work before prepare" instructions.

This invention ensures that loopbacks do not cause unwanted database back sharing. If sharing is attempted from the loopback node, a deadlock will occur to protect the resource manager from corruption.

While the invention has been described with reference to exemplary embodiments, it will be understood by those skilled in the art that various changes may be made and equivalents may be substituted for elements thereof without departing from the scope of the invention. In addition, many modifications may be made to adapt a particular situation or material to the teachings of the invention not be limited to the particular embodiments disclosed for carrying out this invention, but that the invention will include all embodiments falling within the scope of the appended claims.